University Interscholastic League

Computer Science Competition

2013 District 1 Programming Problem Set

DO NOT OPEN THIS PACKET UNTIL INSTRUCTED TO BEGIN!

I. General Notes

1. Do the problems in any order you like. They do not have to be done in order from 1 to 12.

2. All problems have a value of 60 points. Incorrect submissions receive a deduction of 5 points, but may be reworked and resubmitted. Deductions are only included in the team score for problems that are ultimately solved correctly.

3. There is no extraneous input. All input is exactly as specified in the problem. Unless specified by the problem, integer inputs will not have leading zeros. Unless otherwise specified, your program should read to the end of file.

4. Your program should not print extraneous output. Follow the form exactly as given in the problem.

II. Names of Problems

Number	Name
Problem 1	Bears
Problem 2	Dice Roll
Problem 3	Family Tree
Problem 4	Friendly
Problem 5	Lawn Service by Ronnie
Problem 6	Local Warming
Problem 7	Octopusses
Problem 8	Phone Numbers
Problem 9	Qualified
Problem 10	Squares
Problem 11	What's the Difference
Problem 12	X^3

1. Bears

Program Name: Bears.java Input File: bears.dat

Hermit Joe lives in a desolate area of Alaska. The Alaskan Wildlife Management team has given him several maps of the area surrounding his cabin that show the location of the homes of different types of animals. The characters on the map represent different animals that live in the area. For example, the letter E represents an eagle's nest, the letter B represents a brown bear den, the letter M represents a moose habitat, etc. Joe is especially concerned about the brown bears living in the area. You are to write a program that, given a 10x10 rectangular grid of the area, will count the number of brown bear dens. Once Joe has this information, he will know which areas to avoid.

Input

The first line of input will contain a single integer `n` that indicates the number of 10x10 maps to be checked. For each map, there will be ten lines with ten characters and no spaces on each line. All characters will be either an uppercase letter of the alphabet that represents an animal's home or a period (`.`) that represents an area without a major animal home.

Output

For each map, you will print on a single line the number of brown bears living in the area represented by that map.

Example Input File

```
1
E...E..B.M
...B....MM
EECCC.BBMM
..CCCFFBMM
C..F..B..M
B...B..FF.
....FFFFMM
CC..B..MMM
BCCFF...FB
..CC.FF.B.
```

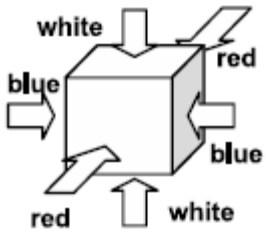
Example Output to Screen

```
12
```

2. Dice Roll

Program Name: DiceRoll.java Input File: diceroll.dat

You are given a single six-sided die as shown on the right. Two of the opposite sides are colored white, two are colored blue, and two are colored red as shown in the picture to the right.



You are to write a simulation that will determine the color that is facing upward after a given number of "rolls". At the beginning of each simulation, the die will be positioned as shown at the right:

- White sides are on the top and bottom.
- Blue sides are on the left and right.
- Red sides are on the front and back.

For each simulation, you will need to construct one object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

For each simulation, you will need two random numbers:

- The first random number `r1` will be a number between 1 and 4, inclusive, and will determine the direction that the die will roll:
  1 = forward, red would be up on the first roll
  2 = backward, red would be up on the first roll
  3 = left, blue would be up on the first roll
  4 = right, blue would be up on the first roll
- The second random number `r2` will be a number between 1 and 25, inclusive, and will determine the number of times the die will roll in the direction determined by `r1`.

Input

The first line of input will contain a single integer `n` that indicates the number of simulations to follow. Each of the following `n` lines will contain a positive `long` which will be the seed for the random object in the simulation.

Output

For each simulation, you will print the color that will be on the top side of the die at the conclusion of the simulation. The color that will be printed will be `RED`, `WHITE`, or `BLUE`.

Example Input File

```
3
12354643225422
546718453
54567755
```

Example Output to Screen

```
WHITE
BLUE
RED
```

Random numbers generated for seeds in the input file:

```
12354643225422
4 20
546718453
3 11
54567755
2 19
```

3. Family Tree

Program Name: Family.java Input File: family.dat

To get some points for a programming contest you decide to write a program that can read in a text file containing a list of familial relationships and parse that data into a family tree. Some data has been lost, however, and you must reconstruct all family links based solely on a list of mother, father and sibling relationships.

Input

The input file will contain an unknown number of lines. Each line will contain 3 words separated by spaces: the source name, the description noun, and the destination name. The description noun describes how the source is related to the destination: `MOTHER`, `FATHER` or `SIBLING`.

Everybody in the tree will have a unique name, and the family tree is guaranteed to be legal in all 50 states. It is possible for the same person to show up multiple times in the input file, but the same relationship between two people will show up only once. You can also assume that any two people in the tree will have a path that joins them.

Output

The output is a generation per line that contains the generation count and the names of every member of that generation in alphabetical order followed by a space. If you were to draw the tree on paper, all people in the top of the tree (the people who don't have any parents in the data) are the 1st generation. Their children are the 2nd generation. The 2nd generation's children are the 3rd generation, and so forth. See output below for an example.

Example Input File

```
Mary MOTHER Bob
Bob SIBLING Gus
Joseph FATHER Gus
Gus FATHER Kim
```

Example Output to Screen

```
1st Generation: Joseph Mary
2nd Generation: Bob Gus
3rd Generation: Kim
```

Note: A space at the end of the line is optional.

4. Friendly

Program Name: Friendly.java Input File: friendly.dat

You and your friends have decided to come up with an encryption method to encrypt text messages that are sent within the group of friends. After a vote the name of the algorithm was decided to be "Friendly" and will be specifically made to encrypt messages of the form "`HH:MM:SS` <text>" where `HH:MM:SS` is the time the message was sent in 24 hour format (with leading 0's whenever a value is below 10) and text is a series of ASCII characters.

The algorithm will work by using 2 bytes (they will be A and B for the purposes of the explanation) which will be XOR'd with the text to encrypt. The timestamp before the text will not be encrypted and be preserved in the resulting string. The algorithm is as follows:

1) Initialize A to equal 9 times the value of the hour from the timestamp.
2) Initialize B to equal 3 times the value of the minutes from the timestamp plus the number of seconds from the timestamp.
3) For each ASCII character in the text portion you will create a new character by XOR'ing the ASCII character with A for even locations in the string and B for odd locations. The first character of the text is location 0.

Input

The input file will contain an unknown number of lines, with one message to encrypt per line.

Output

For each message, you will print the timestamp followed by one space and the encrypted text. The encrypted text will be printed as a space delimited series of hexadecimal numbers representing the byte value of each encrypted ASCII character in the text.

Example Input File

```
01:23:45 Computer science is awesome!
10:11:59 I like ICE CREAM.
```

Example Output to Screen

```
01:23:45 0x4A 0x1D 0x64 0x02 0x7C 0x06 0x6C 0x00 0x29 0x01 0x6A 0x1B 0x6C
0x1C 0x6A 0x17 0x29 0x1B 0x7A 0x52 0x68 0x05 0x6C 0x01 0x66 0x1F 0x6C 0x53
10:11:59 0x13 0x7C 0x36 0x35 0x31 0x39 0x7A 0x15 0x19 0x19 0x7A 0x1F 0x08
0x19 0x1B 0x11 0x74
```

Note: Both of the output lines in the printed version above stretch across two lines, but are both actually only one line in the actual output. Alternate output lines are bolded for ease of reading. A space at the end of each line is optional.

5. Lawn Service by Ronnie

Program Name: Lawn.java Input File: lawn.dat

Ronnie has a lawn service. When figuring an estimate, he needs to know what part of the yard he can mow with his big mower and what part he has to mow with a smaller mower. His big mower covers a 36" square but can only move horizontally or vertically. Any spot in the yard that he cannot get his big mower into, he must mow using the smaller mower. To help him with his estimates, he has made a rectangular matrix of each yard.

Each cell of his matrix represents a one foot by one foot square of the yard that is either an area that needs to be mowed or an area that is not to be mowed. He cannot mow areas where the entry into the area is less than 36" and he may be required to mow over some areas twice (or more). You are to write a program that will show the parts of the yard he can mow with the big mower and the parts he must mow using the smaller mower.

Input

The first line of input will contain a single integer `n` that indicates the number of yards he plans to mow. The first line of each yard will contain two integers `r c` that denote the number of rows and columns respectively in the lawn. The following `r` lines will contain `c` characters with no spaces. The characters are:

- A period (.) denoting an area to be mowed.
- An asterisk (*) denoting an area to mowed around like a tree or flower bed.
- An equal sign (=) denoting the starting cell which will always be in the upper left corner of the matrix.

Output

You will output the matrix for each yard with all periods replaced by:

- A `B` if the area can be mowed by the big mower
- An `s` if the area must be mowed by the small mower.

Print a blank line after each matrix.

Example Input File

```
2
5 15
=.............
...*.....*...*.
...............
...............
......*.....*..
6 20
=....*...*..........
....................
....................
.............*......
...*.............*..
.................*..
```

Example Output to Screen

```
BBBsssssssssss
BBB*sssss*sss*s
BBBBBBssssssssss
BBBBBBssssssssss
BBBBBB*sssss*ss

BBBBB*BBB*BBBBBBBBBB
BBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBB*BBBBBB
BBB*BBBBBBBBBsBBB*ss
BBBsBBBBBBBBBsBBB*ss
```

6. Local Warming

Program Name: Local.java Input File: local.dat

Global warming is an international concern of many people. George wants to compare the months of January in his locality over several select years to see if he can detect a pattern of local warming. He wants to consider the average temperature for each day (the average of the high and low temperature for a given day), and determine the number of days the average temperature was below freezing (32º) and the number of days the average temperature was above 50º.

You have been asked to write a program that will provide this data for him.

Input

The first line of input will contain a single integer `n` that indicates the number of months to follow. Each of the following `n` lines will contain 32 integers. The first integer in each line is the four digit year and each of the following 31 integers represent the average temperature for each of 31 days in January. Each of items will be separated by a space.

Output

For each year input, you will print : `y b w`, where `y` is the four digit year, `b` is the number of days the average temperature was below freezing and `w` is the number of days the average temperature was warmer than 50.

Example Input File

```
3
2011 32 24 16 26 37 48 52 48 57 56 34 23 32 34 23 46 46 56 52 50 43 34 23 28
36 57 59 52 51 50 54
2009 45 54 52 51 58 67 43 32 31 10 18 45 32 21 21 28 29 35 45 54 53 42 51 53
56 44 42 41 42 32 20
2007 54 65 75 74 72 53 52 42 44 41 37 36 39 28 22 25 27 26 45 45 56 57 58 59
63 52 54 63 63 42 12
```

Note: All three of the input lines in the printed version above stretch across two lines, but are all actually only one line each in the `local.dat` input file. Alternate input lines are bolded for ease of reading.

Example Output to Screen

```
2011 7 10
2009 8 10
2007 6 16
```

7. Octopusses

Program Name: Octopusses.java Input File: octopusses.dat

Your boss has used his lack of creativity to create a new game with the grammatically incorrect name of Octopusses and Elevators. The rules of the game are pretty simple:

- Each square on the grid can either have no octopus tentacles nor elevators, 1 to 6 elevators, or 1 to 6 tentacles.
- A given square will never have both elevators and tentacles originating from it.
- A tentacle on a square means you have to slide down along it to a lower numbered square, while an elevator means you get to ride it up to a higher numbered square.
- The starting and ending squares will never have any tentacles or elevators originating from them.

All players start in the lower left corner of a grid. To play the game, each player, in order, takes a turn rolling a 6 sided die with the numbers 1 through 6 on the sides. The player that rolled on a given turn then advances their token the number of squares that corresponds to the number they rolled on the die. The path along the board is shown in the grid below:

64	63	62	61	60	59	58	57
49	50	51	52	53	54	55	56
48	47	46	45	44	43	42	41
33	34	35	36	37	38	39	40
32	31	30	29	28	27	26	25
17	18	19	20	21	22	23	24
16	15	14	13	12	11	10	9
1	2	3	4	5	6	7	8

After a player has moved the number of spaces that are indicated on the die, they then see if the space they landed on has tentacles or elevators.

- If a player lands on a space with the bottom of an elevator, they move up it to a new space.
- If the space has multiple elevators, the player must roll a die to determine which one to take.
    - Each elevator in that space is numbered starting from 0 based on the order they are read in.
    - The player takes the value on the die, subtracts one, and goes along the elevator that matches the value.
    - If the value is larger than the number of elevators then the elevator chosen should be the elevator whose number is the die value - 1 modulus the number of elevators. For example, if the die value is 4 and the number of elevators is 2, then the elevator chosen is number 1 (the second elevator), i.e., (4-1) mod 2 =1.
    - This same logic applies to the tentacles, numbered starting from 0 based on the order they are read in.
- Once the player follows an elevator or tentacle and the new space then has an elevator or tentacle, the player performs the same logic again, rolling a die if needed, and continues doing so until he gets to a space where he can no longer move along an elevator or tentacle.
- The game is won when a player gets to or passes the box labeled 64.

To simulate the dice rolls, you should construct an object of the type `java.util.Random`. This class allows you to specify the seed for the random number generator. For a given seed, the order of the random numbers is always the same.

Input (cont. on next page)

7. Octopusses (cont.)

Input

The first line will contain a single integer `n` that indicates the number of games to follow. For each game:

- the first line will contain a single integer that indicates the number of players in the game.
- the second line will contain the seed; a single `long` integer.
- the third line will contain 2 integers `t e`, where `t` is number of tentacles on the board and `e` is the number of elevators.
- Each of the next `t` lines will contain two integers `b s`, where `b` is the square where the tentacle begins and `s` is the square where the tentacle stops.
- Each of the next `e` lines will contain two integers `b s`, where `b` is the square where the elevator begins and `s` is the square where the elevator ends.

Output

For each game you print a single line which reads: `Player X wins after Y rolls!` Each player is given an uppercase character, with the first player being `A`, the second being `B`, etc. *`X`* is the character corresponding to the winning player and *`Y`* the number of rolls that happened during the game, including the winning roll and rolls needed to determine which path to take. If for example there are two players, and player A rolled 6 times and player B rolled 5, and player A's 6th roll won the game, then the string would read `Player A wins after 11 rolls!`

Example Input File

```
1
2
77583464
8 7
27 10
27 9
27 1
10 6
10 9
57 41
34 14
34 11
14 37
14 45
17 49
50 64
50 62
22 37
22 23
```

Example Output to Screen

```
Player B wins after 21 rolls!
```

Random numbers generated for given seed:

```
77583464
5 3 4 4 3 6 6 4 4 4 5 6 5 2 1 3 5 4 3 4 3
```

8. Phone Numbers

Program Name: PhoneNumbers.java Input File: phonenumbers.dat

Your team has been assigned to write an app that obtains personal information from the users of the app. You have been assigned the part of the program that verifies the user's phone number.

The user is told to enter their phone number in the form `ddd-ddd-dddd` where `d` is a digit, 0 through 9. You are to verify that the user's input is correct.

Input
The first line of input will contain a single integer `n` that indicates the number of phone numbers to follow. Each of the following `n` lines will contain a single string with no spaces.

Output
For each string input, you will print the phone number if it is formatted correctly. If the phone number is not formatted correctly, you will print: `INVALID PHONE NUMBER`.

Example Input File
```
4
214-234-5647
456.765.3456
(768)567-4536
342-3098
```

Example Output to Screen
```
214-234-5647
INVALID PHONE NUMBER
INVALID PHONE NUMBER
INVALID PHONE NUMBER
```

9. Qualified

Program Name: Qualified.java Input File: qualified.dat

Roger is sponsoring an elite programming contest at his school. In order to ensure the quality of the contestants, he has decided to require a qualifying score to enter. You are to write a program that will determine which teams have qualified to enter his contest.

He has decided that to qualify for his contest, the programming team must have previously competed at another recognized contest. Since scores at different contests are not uniform, he has decided that some contests should require a higher minimum score than others. He has decided on the following:

Contest	Abbreviation	Qualifying Score
Southeast Texas Championships	STC	560
West Texas Championships	WTC	340
North Texas Championships	NTC	420
Central Texas Championships	CTC	370
Lower Valley Championships	LVC	470

You are to write a program that will print the names of the teams qualified to enter his contest.

Input
The first line of input will contain a single integer `n` that indicates the number of teams wanting to enter his contest. Each of the next `n` lines will contain a team name with no spaces followed by a space, the team score from a recognized contest, a space, and the three letter abbreviation of the recognized contest.

Output
In the order they appear in the original list, you will print the names of the teams qualified for the elite programming contest.

Example Input File
```
8
PASCAL_HS 455 WTC
TURING_HS 365 CTC
WIRTH_HS 570 STC
WASHINGTON_HS 500 LVC
ADAMS_HS 325 WTC
ROBINSON_HS 480 NTC
GATES_HS 400 CTC
JOBS_HS 450 LVC
```

Example Output to Screen
```
PASCAL_HS
WIRTH_HS
WASHINGTON_HS
ROBINSON_HS
GATES_HS
```

10. Squares

Program Name: Squares.java Input File: squares.dat

For a project your class is doing, you have been assigned to generate a set of concentric squares.

Input
The only line of input will contain an unknown number of positive integers that indicate the size of the largest square in the designs that you are to create.

Output
For each design, you will print a square with ***x*** asterisks (*) on each side. Then inside that square will be another square, one cell away on each side. This pattern will continue until there is a single square in the middle, thus making a box of concentric squares. Print at least one blank line after each design.

Example Input File
```
8 9
```

Example Output to Screen
```
********
*      *
* **** *
* *  * *
* *  * *
* **** *
*      *
********

*********
*       *
* ***** *
* *   * *
* * * * *
* *   * *
* ***** *
*       *
*********
```

11. What's the Difference

Program Name: Whats.java Input File: whats.dat

James owns a music store that he keeps open seven days a week. He would like to close one day a week to spend more time with his family. In an effort to determine which day would be the best for him to close his business, he has collected the statistics on his sales for each day for four weeks plus a day 0, the day before the four week period began.

You are to write a program that will find the four days over the interval day 1 through day 28 which have the least difference in sales from the preceding day. For example, if day 0 has $1300 in sales, day 1 has $1500 in sales, and day 2 has $1200 in sales, then day 1 has a difference of $200 and day 2 has a difference of -$300.

Input

The first line of input will contain a single integer n that indicates the number of lines of data to follow. Each of the following n lines will contain 29 integer values, separated by a space, representing James' sales on days 0 through 28.

Note: You may assume there will be no ties in the four smallest differences.

Output

In order from least to greatest, you will print the four smallest differences and the day number of that difference with no dollar ($) signs. Separate the difference and the day number by a space. Print at least one blank line after output sets.

Example Input File

```
2
1300 1500 1200 1600 1800 900 1200 1200 1400 1500 1600 1200 1450 1475 1525
1600 1125 1200 1365 1545 1900 1325 975 1300 1445 1235 1534 1534 1200
1745 2034 2000 2100 2100 1975 1823 2231 1992 2000 2345 2435 1982 2400 1934
1823 2354 2000 2137 2579 1900 2336 2643 1745 2343 2155 2354 2789 2945
```

Note: Both of the input lines in the printed version above stretch across two lines, but are all actually only one line each in the `whats.dat` input file. The second input line is bolded for ease of reading.

Example Output to Screen

```
-900 5
-575 21
-475 16
-400 11

-898 23
-679 20
-466 14
-453 12
```

12. X^3

Program Name: X3.java Input File: x3.dat

Ms. Appleworth is a mathematics teacher and is teaching his students about cubic equations. He needs you to find the integer solutions to some cubic equations for him. The equations are in the form $ax^3 + bx^2 + cx + d = 0$. He knows all of the solutions will be in the range [-100 100]. He has asked you to write a program that will find these solutions for him.

Input
The first line of input will contain a single integer `n` that indicates the number of equations to follow. Each of the following `n` lines will contain four integers in the form `a b c d` representing the values of the coefficients a, b, c, and d respectively of the equation that is in the form of the equation above. Each of items will be separated by a space.

Output
For each equation, you will list, on a single line and folled by a space, the integer solutions to the equation. A space at the end of the line is optional. If no solution in the above interval exists, print `NO INTEGER SOLUTION`.

Example Input File
```
3
1 2 3 2
1 -3 -4 2
1 3 -10 -24
```

Example Output to Screen
```
-1
NO INTEGER SOLUTION
-4 -2 3
```